Hindawi
Mobile Information Systems
Volume 2022, Article ID 3299592, 10 pages
https://doi.org/10.1155/2022/3299592

Research Article

Application of Knowledge Model in Dance Teaching Based on Wearable Device Based on Deep Learning

Yan Zhang

Northeast Dianli University, Jilin 132001, China

Correspondence should be addressed to Yan Zhang; 160407109@stu.cuz.edu.cn

Received 9 May 2022; Accepted 21 June 2022; Published 8 August 2022

Academic Editor: Muhammad Muzammal

With the great changes in the social, economic, and cultural background, the traditional education model can no longer meet the current goal of cultivating dance talents. The application of wearable devices based on deep learning helps to improve students' understanding and application of dance movements. Through extensive data analysis, experimental research, and kinesthetic theory, it is found that trainers based on deep neural networks can effectively improve students' overall learning performance. At the same time, the application of emotion-intelligence teaching mode theory in dance experiments and the conclusions drawn from the experimental research fully demonstrate the teaching advantages of applying deep learning wearable devices to dance emotion-intelligence teaching mode. In order to explore the application of wearable devices based on deep learning in dance teaching, this paper discusses the improvement of deep learning through the elaboration of parameter smoothing initialization, convolutional pooling layer, optimal smoothing filter, and dynamic pruning method, and then a wearable device is designed. A device can recognize dance movements to verify the application of deep learning-based wearables in dance teaching in emotional mode.

1. Introduction

With the development and maturity of new technologies, there is an increasing demand for the use of intelligent terminals. Deep neural networks are highly fault-tolerant and can implement complex system models describing various information to meet the functional requirements of different user groups. They process large amounts of data and integrate it into a machine that provides multiple solutions and multiple types of services to help humans solve problems, and are well adapted to various types of reasoning methods that do not rely on artificial models.

Deep learning is a new artificial intelligence tool based on the Nans bus technology for digital image processing. It is based on parallel computing and uses deep neural networks and multilayer perceptrons to extract information from complex features [1]. As there are many uncertainties affecting human cognitive activities, e.g., environmental noise, ambiguous stimuli, etc., can lead to interference or even failure in the machine recognition process resulting in erroneous judgment results; in order to solve these problems, we introduced deep learning techniques to analyze a large number of raw images captured in real environments and then obtain information on the attributes of the target objects that need to be interpreted and predicted and then use. This useful information is then used to achieve description, recognition, or tracking tasks [2].

Research on wearable devices is mainly focused on intelligent robots and embedded systems. Currently, machine vision and neural networks based on deep learning are widely used in industrial production processes. There are two main R&D techniques: one is based on neural networks to establish ubiquitous layer bit algorithms, deep learning models, and parallel underlying computing systems, and the second is terminal signal strength. Due to the complexity of deep learning, we need to address the problems that exist in the current environment when implementing wearable devices [3]. Current existing research focuses on visual, auditory, and haptic aspects. In terms of vision, the first priority is to improve the system's perceptual capabilities, the second is to increase real time, and the last is to enhance the sense of movement and improve features such as motion performance to further enhance the interaction experience and stability. Neural networks are used in a wide variety of applications due to their high adaptive capacity and robustness [4] and are used in a wide range of applications, including in dance teaching.

The effect of full concatenation on the neural network can be achieved by using global average pooling in place of the full concatenation layer for the purpose of mapping a two-dimensional feature map to a one-dimensional vector. The main idea of global average pooling is to do a full-size average pooling of the feature maps generated by each channel of the final convolutional layer in the convolutional neural network to obtain a value [5–7], and the values of all channels are weighted and averaged as the final score for that sample.

2. Improved Deep Learning Methods

2.1. Smooth Initialization of Parameters. Since the training process only amounts to smoothing most of the convolutional kernels once, what would happen if we smoothed the convolutional kernels ahead of time between the start of the network training? Does this save the resources wasted on "smoothing" during the training process and thus improve the accuracy of the network? All parameters are still initialized according to the Xavier method, all convolutional kernels are smoothed using a 3×3 Gaussian filter to obtain a set of smooth convolutional kernels, and the smoothed network is trained for the same period using the momentum method with a step size of 0.01 and a momentum factor of 0.5.

First consider a basic shallow network. The structure of this network is shown in Figure 1. The first convolutional layer of the network considered here contains 8 neurons, the second convolutional layer contains 16 neurons, and the first fully connected layer contains 64 channels. All convolutional kernels are of size 5×5. The network contains a total of 20522 parameters.

Similarly, we trained and classified the MNIST dataset. All parameters within this network are initialized using the Xavier method (equations (1) and (2)). The gradient descent method is momentum (equation (3)) where the step size is 0.01 and the momentum coefficient is 0.5. The activation function of the hidden layer is ReLU (equation (4)), and the activation function of the output layer is a softmax function. We choose cross-entropy (equation (5)) as the error function.

$$N\left(0, \sigma^2\right), \sigma = \sqrt{\frac{2}{\left(c_{in} + c_{out}\right)}}, \tag{1}$$

$$U(-a, a), a = \sqrt{\frac{6}{\left(c_{in} + c_{out}\right)}}, \tag{2}$$

$$v_{t+1} = \varepsilon v_t - \mu \nabla f\left(\theta_t\right) \theta_{t+1} = \theta_t + v_{t+1}, \tag{3}$$

$$\begin{aligned} L_{\text{relu}}(x) &= x^{+}, \\ &= \max(0, x), \end{aligned} \tag{4}$$

$$H(p, q) = -\sum_i p_i \log q_i. \tag{5}$$

After 9370 iterations (10 cycles) on the MNIST training set, our small network achieves a test accuracy of 98.77%. It should be noted that since the MNIST training set contains 60,000 samples, we use a small batch approach in the training process, i.e., 64 samples per iteration, so that after 937 iterations, all training samples are traversed once. In order to extend the training time and further improve the accuracy of the network, we iterate through all the samples once (called one cycle, epoch); i.e., all the training samples are randomly ordered and iterated through again. If we visualize the initial trial state of the convolutional kernels versus the state at the end of training, we see that there are many convolutional kernels that do not change significantly from their initial state, and most of the trained kernels are actually only smoothed out a bit from their initial values. To test whether smoothing the convolutional kernels in advance between the start of network training could improve the accuracy of the network, I designed the following experiments: firstly, the structure of the shallow network did not change and all parameters were still initialized according to the Xavier method. Secondly, all the convolutional kernels were smoothed using a 3×3 Gaussian filter to obtain a set of smooth convolutional kernels. Finally, the smoothed network was trained for the same period using the same step size of 0.01 and momentum factor of 0.5 momentum. We refer to the smoothing operation added after the initialization of the network parameters and before the training starts as the smooth initialization. Accordingly, a smoothed convolutional neural network is called a smooth convolutional neural network. We can see that the smoothed set of convolutional kernels splits into two subsets at the end of training, with a small portion of the convolutional kernels having more significant pixel variation and a larger standard deviation, and a large portion of the convolutional kernels becoming smoother and having a smaller standard deviation. At this point, it is reasonable to assume that the convolutional kernel with the larger standard deviation plays a more dominant role in the classification process, as the kernels with their own distinctive features will also be able to extract more salient features and thus obtain more useful texture information in the original data. Convolutional kernels with small standard deviations play a minimal role in the classification process, as they tend to have uniform pixel values and their output values are only influenced by the input values, regardless of the input data, and do not provide effective feature information to the neurons in the later layers. After 10 cycles of training, the test accuracy of the smooth network was almost identical to that of the non-smooth network (98.72%). Combined with the training results for the fully trained smooth network with fully connected layer parameters, we found that there is actually a degree of diversity required in the fully connected layer. This

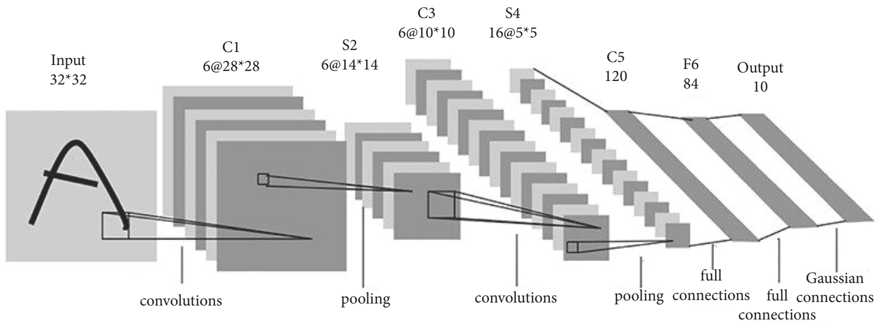

Figure 1: MNIST classification of CNN structure.

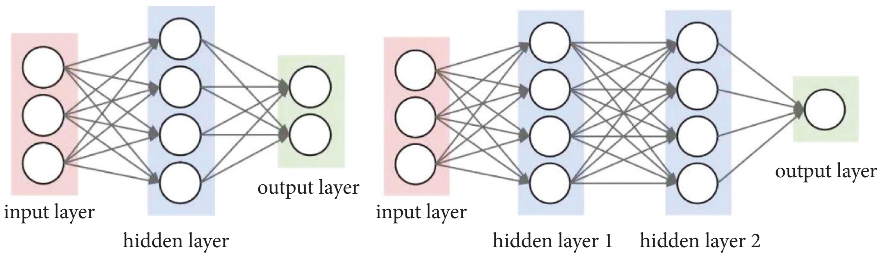

Figure 2: Typical fully connected layer.

suggests that performing a direct smooth initialization of the fully connected layer is not ideal and that we need to find other ways to initialize the parameters of the fully connected layer so that it has both a smooth and a highly variable part.

2.2. Convolutional Pooling Layer. A convolutional pooling layer is essentially still a convolutional layer, except that the size of its convolutional kernel is the same as the size of the input data, and the number of output channels (like a fully concatenated layer) is an arbitrary positive integer. Since the convolutional kernel is the same size as the input data, the result of each convolutional operation is a single value. Thus, the convolutional pooling layer's output can be directly constituted as a vector. In fact, the C5 layer in LeNet-5 [8] is a convolutional pooling layer with a 5×5 convolutional kernel and input data. Compared to a fully connected layer, the number of parameters and the amount of computation in a convolutional pooling layer are the same for the same number of output channels. As shown in Figure 2, a fully connected layer with input channel k and output channel c has an input feature map of size $w - h$ (w and h denote the width and height of the feature map, respectively). The parameters W of the fully connected layer contain $k - c - w - h$. This is split and reorganized into k-c matrices of size $w - h$, each of which can be used as the convolutional kernel of the convolutional pooling layer. The number of parameters of the convolutional pooling layer is also $k - c - w - h$. In order to investigate the compatibility of the convolutional pooling layer with the smooth initialization method proposed in the previous section, I designed the following experiment. The first fully connected layer in the shallow neural network is replaced by a convolutional pooling layer with the same number of channels, and the other parameter settings are kept unchanged. We named the network as convolutional pooling network for ease of differentiation. Similarly, we compared the performance of the convolutional pooling network between nonsmooth and smooth initialization and designed the following experiments for different kinds of layers in the network: training only the convolutional layers of the nonsmooth network versus the smooth network; training only fully connected layers (corresponding to "fully connected"); and training both convolutional and fully connected layers (corresponding to "all layers"). It is important to note that all-connected includes the traditional all-connected layer, the convolutional pooling layer, and of course the output layer (which is actually an all-connected layer in its own right).

From Figure 3, it can be seen that the network with both the convolutional pooling layer and the smooth initialization achieves some improvement in accuracy (98.90%) over the original network (98.77%). Further comparison shows that the convolutional pooling network (with smooth initialization) is more accurate than the original network when only the convolutional layers are trained. This indicates that the convolutional pooling layer outperforms the fully connected layer even when there is no training. In addition, the standard deviation of all accuracies for the convolutional pooling network (0.71) is lower than that of the original network (0.74). This indicates that the convolutional pooling layer makes the overall performance of the neural network more stable.

2.3. Optimal Smoothing Filter. In addition, we compared the performance of other smoothing filters with the 3×3 Gaussian filter to analyze the different filters on the parameter distribution to determine the optimal smoothing

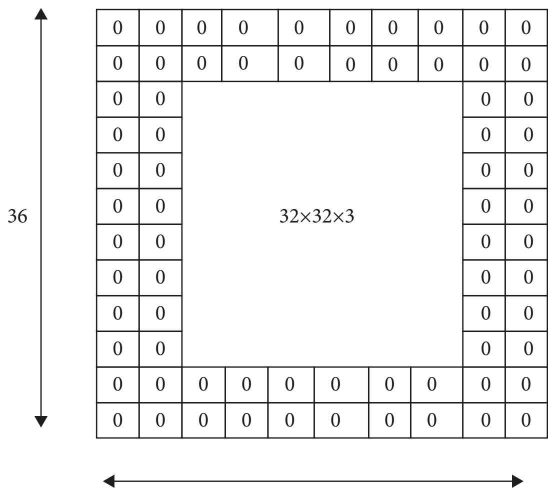

Figure 3: The convolutional pooling layer and the fully connected layer are equivalent after parameter restructuring.

filter for smooth initialization. Specifically, we compared the performance of a 3×3 Gaussian filter, a 5×5 Gaussian filter, a 3×3 median filter, and a 3×3 median filter with a non-smooth initialization. To make the comparison more convincing, we trained and tested on the SVHN dataset, a real-world digital sample set extracted from Google Street View images, which contains 73257 training samples, 26032 test samples, and 531131 slightly simpler additional training samples. We combined the training samples with the additional training samples into one training set and used a three-layer convolutional neural network for the experiments. The three hidden layers of the network contain 64, 96, and 128 channels, respectively, and the flattening layer uses a convolutional pooling layer with 2048 channels. The network contains over 4 million parameters in total. We use momentum (equation (4)) as the training method, where the momentum factor is set to 0.5. The initial training step is set to 0.1 and is scaled down by a factor of 0.1 every 2 cycles starting from the 5th training cycle until the end of the 10-cycle training. The parameters of the network were initialized with the Xavier method, followed by a smooth initialization. ReLU (equation (3)) is used as the activation function in the hidden layer, while softmax is used as the activation function in the output layer and cross-entropy (equation (5)) is used as the error function. Different smoothing filters result in different distributions of parameters, which in turn are reflected in different accuracy performances of the neural network. In this paper, we choose a 3×3 Gaussian filter as the default filter for the smoothing initialization.

2.4. Dynamic Pruning Methods. The first is pruning of flattened layers, by virtue of convolutional pooling layers, which allows us to treat most fully connected layers as convolutional layers. The advantage of this is that we can train and prune the flattened layer in the same way as the convolutional layer, unlike either treating the convolutional layer differently from the fully connected layer in the pruning process or just sparse the convolutional parameters

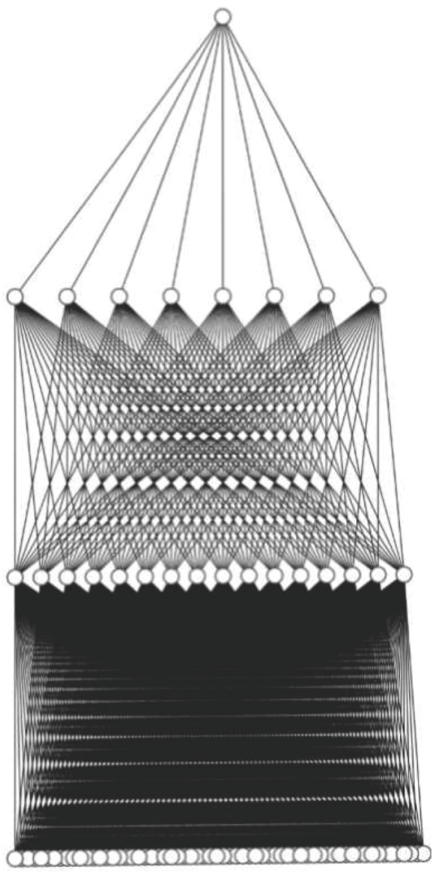

Figure 4: Before network pruning.

without treating the fully connected layer. It is important to note that we do not prune the fully connected layer after the flattening layer—more specifically, the fully connected layer that is itself the output layer of the network. The reasons for this are twofold: our aim is not to compress the storage space occupied by network parameters to the smallest possible proportion, and moreover, the number of parameters at this layer is generally not very large in itself. We believe that this layer is responsible for linearly transforming the results obtained by abstracting the data from all previous hidden layers to serve as the basis for final decisions such as classification, and it has been experimentally demonstrated that the parameters in this layer generally obey a nonsparse distribution and that the differences between adjacent values are relatively large. Therefore, the redundant parameters at such a high-level position in the network are relatively few, and there is no need to prune the parameters of this layer.

In addition, the fully connected layer after the convolutional pooling layer can be replaced by a convolutional layer with a kernel size of 1×1, so that the whole convolutional network can be considered as designed in a "full convolutional" way. Since we do not prune the fully connected layers after the convolutional pooling layer, we do not distinguish between these two forms of computation, but refer to them all as "fully connected layers."

The second is neuronal degeneration. After removing unimportant parameters from a neural network, it is

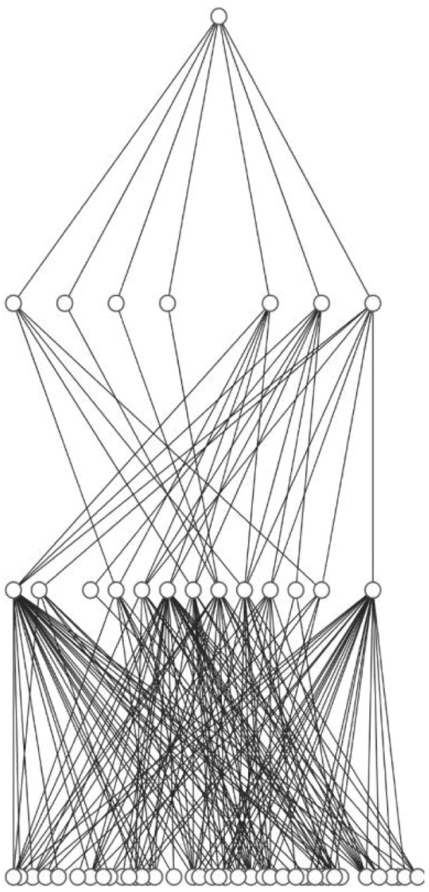

Figure 5: Network before pruning.

sometimes necessary to continue removing unnecessary neurons (nodes). As a result of parameter pruning, some hidden layer neurons have no input connections at all or no output connections. This makes these neurons detached from the decision-making process of the network, so deletion is needed to further simplify the structure of the network. We refer to this process of removing unnecessary neurons as neuronal degeneration. Neuronal degeneration is also a simulation of the process of neuronal apoptosis in the mammalian brain. As shown in Figure 4, the combination of parameter pruning and neuronal degeneration results in a more concise and clearer structure of the neural network. It is important to note that there is a relatively rare scenario in neuronal degeneration: when a cryptic neuron has only one input (or output) neuron, the neuron degeneration may result in a stranded neuron. In Figure 5, neuron A has one input connection and one output connection, while its output neuron B has only one input connection and no output connection. According to the neuron degeneration rules, neuron B is deleted and neuron A is retained. However, when neuron B is deleted, neuron A has only one input connection and no output connection (as in Figure 6). At this point, neuron A is still not involved in the decision computation of the neural network. We call neuron A a stranded neuron. To solve the problem of stranded neurons, we only need to perform another neuron degradation process on the network. Specifically, for a neural network with k hidden layers, we perform k neuron degenerations to ensure that no stranded neurons are created. Also, to save computational resources, we perform only one neuron degradation during the training process if necessary and only perform k neuron degradations after the training is completed.

A is still not involved in the decision computation of the neural network. We call neuron A a stranded neuron. In order to solve the problem of stranded neurons, we only need to perform one more neuron degeneration on the network. Specifically, for a neural network with k hidden layers, we perform k neuron degenerations to ensure that no stranded neurons are created. In addition, to save computational resources, we perform only one neuron degradation during the training process if necessary and only perform k neuron degradations after the training is completed.

To illustrate the network pruning process in detail, we take the example of a three-recessive layer convolutional neural network trained for the SVHN dataset. We know that the median filter produces a blocky appearance on the image, and that choosing the median within a neighborhood inevitably discards the extreme values, which results in the median filter producing parameters with smaller magnitudes and smaller standard deviations. According to this analysis, more parameters should be pruned in a network with a median filter. However, although the parameters generated by the median filter have a small standard deviation at initialization, during the training of the network they become even larger than those of the Gaussian filter [9–18].

3. Experimental Simulation and Analysis

3.1. System Development Environment. In this paper, unlike the client and server architecture used in the previous experiments, the system transplants the above algorithm and uses a wearable smartwatch to realize the recognition of subtle human hand movements to, mainly working on real-time data collection and processing, extraction of movement fragments, real-time classification of hand movement types, and output of meticulous movement recognition results on the smartwatch screen. Wearable device hardware: Considering that the current mature commercial Android smartwatch devices have various kinds of built-in smart sensors and provide secondary development interfaces, after screening and research, this paper adopts the Huawei Watch 2 smartwatch for hardware device development, which has built-in acceleration and gyroscope sensors and has good computing capability to meet the system hardware. It has built-in acceleration and gyroscope sensors and has good computing power to meet the system hardware development requirements. In the implementation of the system, this paper uses Android Studio to develop and port the motion recognition system in Windows system.

(i) Hard Disk: 1000 GB

(ii) CPU: Intel Core i5-4590CPU @3.30 GHz

(iii) RAM: 8 GB

(iv) OS: Windows 8.1 Professional Edition

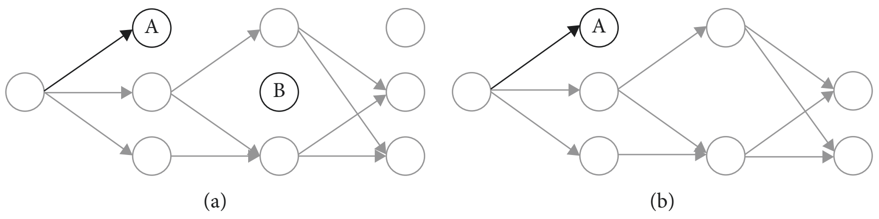

Figure 6: Stranded neurons. (a) Neuron B is degraded and A is retained; (b) neuron B is degraded and A becomes a stranded neuron.

(v) IDE: AndroidStudio2.2

(vi) Java SDK: jdk1.8.0_65

(vii) Android SDK: Android 4.4.4

3.2. System Implementation and Functional Modules. In the system implementation part, the smartwatch hardware mainly implements real-time data acquisition, saves it in the local file of the smartwatch to provide data support for the next data preprocessing operation, and segments the continuous movements to extract the complete fragments of the movements, and the smartwatch distinguishes different hand movement types according to the motion energy of the calculated movement fragments and, furthermore, extracts the corresponding feature values to input into the classification model. Further, the corresponding feature values are extracted and fed into the classification model to classify the categories of meticulous human hand movements in real time, and finally, the output is displayed on the smartwatch. In the implementation part of the system, there are two models to choose from: the experimental analysis model and the systematic recognition model. The experimental analysis model is responsible for the analysis and processing of the offline data on MATLAB, the verification of the feasibility of the relevant algorithms, the construction of the feature vector sample data, and the construction of the classifier model using the Weka environment; the system recognition model, based on the previous work, transposes the recognition algorithm into the smartwatch, recognizes the subtle human hand movements in real time, and displays the results on the screen. The above part of this paper has detailed the relevant experimental and research part of the offline recognition solution; this chapter will make a detailed description of the real-time part of the system. Based on the functional analysis, this solution combines the process of human arm minutiae movement recognition and the overall flow of the system divides the system into four broad modules according to their respective functions, including the data processing module, the classification of hand movement types, the recognition of minutiae movements, and the output module.

3.3. System Module Description and Implementation. In order to enable the system to effectively recognize the subtle movements of each hand, the smartwatch first needs to carry out the necessary preprocessing of the data. The data processing mainly consists of data acquisition and data preprocessing, where the data acquisition module is responsible for collecting the user's behavioral data and saving it in the smartwatch file, the data.

The data collection module is responsible for collecting the user's behavioral data and saving it in the smartwatch file; the data preprocessing module mainly contains the work of raw data filtering and noise reduction.

Data acquisition. To facilitate the collection of human hand movement data in real life without affecting the user's daily behavior, the system uses a commercial Huawei Watch 2 smartwatch with a wealth of built-in sensors. When the user makes a relevant hand movement while wearing the device (the user wears it in the right hand), the system will collect and cache the raw data of the hand movement. The implementation of the system data acquisition module consists of the following two main steps:

(1) Motion sensor acquisition: We first define the accelerometer and gyroscope sensors and use the SensorManager manager to obtain, their instance objects are obtained using the SensorManager, and operations such as fetching and releasing are performed on them. The code is shown below.

```
//Define the sensors
private SensorManager mgr;
private Sensor accel; //accelerometer
private Sensor gyro; //gyroscope sensor
//Get the sensor
mgr = (SensorManager) this.getSystemService
(SENSOR_SERVICE);
accel = mgr.getDefault
Sensor(Sensor.TYPE_ACCELEROMETER);
gyro = mgr.getDefaultSensor
(Sensor.TYPE_GYROSCOPE)
//release the sensor
mgr.unregisterListener(this, accel);
mgr.unregisterListener(this, gyro);
```

(2) Acquisition of sensor data: Each sensor acquisition principle is the same; here, only the gyrosensor acquisition data link is described. Before collecting the data, the gyroscope registerListener listener is registered with the sensor manager, which listens for changes in the sensor data and is output by event.values[i]. The code for this is as follows.

```
mgr.registerListener(this, gyro, SensorManager.SENSOR_DELAY_GAME); //set the sampling frequency
switch (event.sensor.getType()){
case Sensor.TYPE_GYROSCOPE://Gyroscope sensor determination
numberofGyro ++;
for(int i = 0; i < 3; i++) {
gyroData[i] = event.values[i]; }//output angle in radians per second
```

When the dance is learned and the human hand is in motion, the sensor data will be cached in the BufferedWriter, and when the human hand is finished, the data will be written to the smartwatch file when the END button is clicked. The raw data file captured by the smartwatch is full of burrs and noise and cannot be used directly to extract and classify hand movements, so it needs to be cleaned and transformed into a valid hand movement sensing data. When performing the acceleration data acquisition, considering that the sensor's native output is affected by the acceleration of gravity and the linear acceleration generated by human behavior, this paper has filtered it through a low-pass filtering algorithm to obtain the acceleration data generated by gravity and movement, respectively.

```
//Gravitational acceleration and motion-generated acceleration data are obtained separately from the accelerometer
for(int i = 0; i < 3; i++) {
gravityData[i] = (float) (0.2 * event.values[i] + 0.8 * gravityData [i]);//gravity acceleration data. motionData [i] = event.values[i]-gravityData[i]; //acceleration data from motion
...}
```

At the same time, we use a moving-mean filtering algorithm to remove burrs and noise from the raw data, smoothing the signal.

This smoothes out the noise and burr jitter and removes irrelevant details from the raw data.

```
////moving-mean-filter denoising
int n = 8;
double[] acc_squar_avg = new double[size_acc -n + 1 + size_acc]; for (int i = 0; i < size_acc -n + 1; i++) {
acc_squar_avg[i] = getArraySum(acc_squar, n, i)/n;}
for (int i = size_acc -n + 1; i < size_acc; i++) {
acc_squar_avg[i] = acc_squar[i]; }
```

In order to obtain the human hand action fragments, this paper obtains the complete hand action fragments through the initial detection of the hand action data model and the improvement of the adaptive action fragment extraction algorithm. The process is specifically divided into the following three steps:

(1) Initial detection of the hand action fragment model: We use a sliding window to divide the continuous data sequence into several equal parts, and set the size of the sliding window to 10.

Save the position of the candidate points of the action fragment model to obtain the preliminary hand action fragment model.

```
/* hand action fragment model preliminary detection algorithm */
int spliding_window_size = 10; //set the length of the sliding window
double[] peaks_pos = falses(size_acc); //save the obtained peak candidate positions
for (int i = 0; i < size_acc; i = i + spliding_window_size) {
if (Math.floor((double) i/spliding_window_size) ! = Math.floor((double) size_acc /
spliding_window_size)) {
for (int j = 1; j ≤ spliding_window_size; j++) {
temp_pos[j−1] = acc_squar_avg[i + j − 1]; }
double temp_avg = mean(temp_pos); //calculate the mean value
//empirical threshold
if (temp_avg >Mean_ threshold) {
for (int j = 1; j ≤ spliding_window_size; j++) {
peaks_pos[i + j − 1] = 1; //candidate peaks position
int[] temp_start = new int[segment_count];
int[] temp_end = new int[segment_count];
int temp_count = 0; for (int i = 1; i < peaks_pos.length; i++) {
if ((peaks_pos[i] ! = peaks_pos[i − 1]) && (peaks_pos[i] = = 1)) {
temp_start[temp_count] = i + 1;
segment_count++; }
if ((peaks_pos[i] ! = peaks_pos[i − 1]) && (peaks_pos[i] = = 0)) {
temp_end[temp_count] = i;
temp_count = temp_count + 1; }
}}
...
```

(2) Detecting the start point estimate and end point estimate of the action fragment: Extend the TEMP_N data sampling points to both sides of the above detected start and end points.

To obtain the segment_start_estimate and the segment_end_estimate of the hand movement segment.

segment_end_estimate

```
/* Adaptive motion segment extraction improvement algorithm to detect start and end estimates */
int[] temp_start; //record the start point of the above action segment preliminary model
```

int[] temp_end;//record the end point of the initial model of the above action fragment

double[][] temp_look = zeros(Math.ceil(size_acc/spliding_window_size), 3);//*r* keep the results of each window

The results of the calculations within each window, Var, Avg, and Diff

int[] segment_start_estimate = new int[segment_-count]; //; //record the estimate of the start point of the action segment

int[] segment_end_ estimate = new int[segment_-count]; ////record the action segment end point estimate

int temp_N = 25; ////initial probe point extends N data points in size to both sides

////Action fragment peak detection algorithm start point estimate

for (int $i = 0$; $i <$ segment_count; i++) {

int j = temp_start[i]; ///the value of the change in the termination point of the action segment size, with an initial value of 1

while ($j > 0$ && acc_squar_avg[$j - 1$] > accel_terminal_threshold) {

$j = j - 1$; }

int k = j-temp_N > 0 ? j-temp_N: 0;//// action_segment_start_valuation

segment_start_ estimate [segmegt_start_temp] = k;

segmegt_start_temp = segmegt_start_temp + 1; }

////action snippet peak termination point estimate of the detection algorithm

for (int $i = 0$; $i <$ segment_count; i++) {

....;

}

(3) Accurate extraction of complete action segments: A sliding window of length 5 is set up, sliding backward from the estimated starting point of the extension and sliding forward from the estimated ending point of the extension, and calculating the variance Var, Mean, and Diff of the data sampling points in the sliding window. If the three eigenvalues are greater than the threshold, then the data points in the sliding window are judged to be on an upward trend and the sliding is stopped. In any case, the sliding of a data point continues, and the final detection of the action fragment's state point precision value.

/ ∗ Adaptive action fragment extraction improvement algorithm, detecting the exact value of the start point and the exact value of the end point ∗ /

int temp_windos = 5; ////sliding window length

int[] segment_start = new int[segment_count]; ////array of record exact start points

int[] segment_end = new int[segment_count]; ////record the exact end point array

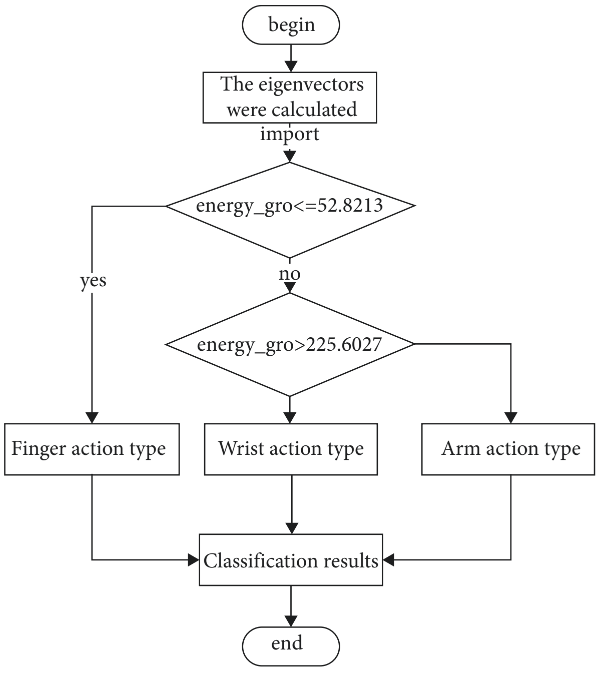

Figure 7: Flow of hand action-type classification.

///Pick up the exact position of the start point based on the variance Var, mean, and difference Diff thresholds

for (int $i = 0$; $i <$ segment_count; i++) {

int start_estimate_point = segment_start_estimate[i];

for (int j = start_estimate_point-1; $j <$ size_acc; j++) {

temp_var = var(acc_squar_avg(j:j + temp_windos-1)); ////////calculate variance

temp_mean = mean(acc_squar_avg(j: j + temp_windos-1)); ////////calculate the mean

double temp_diff = sum_diff(acc_squar_avg, j, j + temp_windos-1); ///////calculate sum of differences

//compare the magnitude of the variance Var, mean Mean, and sum of differences Diff with the threshold within the sliding window

if (temp_diff > diff_thred_start&& temp_var > var_thd&& temp_mean > mean_thd) {

segment_start[temp_start_accurate-1] = j + 1;

temp_start_accurate = temp_start_accurate + 1;

break; }

} }

///Pick up the exact position of the termination point based on the variance Var, mean Mean, and difference Diff thresholds

for (int $i = 0$; $i <$ segment_count; i++) {

int end_estimate_point = segment_end_estimate[i];

while (end_estimate_point > temp_end[i]) {

... }

}

In order to classify the hand movement types of dancers, we need to differentiate between arm, wrist, and finger movement types by means of a classifier. We used a decision tree model to classify the hand movement types. We constructed a training sample set from all the hand movements and obtained the results for the three hand movement types (arm, wrist, and finger). In the implementation process, the decision tree classification model is used to calculate the energy values in the motion segments to differentiate the different hand movement types. In the implementation process, the decision tree classification model is used to calculate the energy values in the motion segments to differentiate the different hand movement types (see Figure 7).

In this paper, the ClassifyHandType function is used to calculate the energy of the gyroscope data in the motion fragment energy_gro and finally obtains the classification result of the hand action type.

In a real-world environment, the test sample set TD was used as input to the system to obtain real test results, which consisted of two parts. The first part is the accuracy of the recognition of human hand movement types (arm, wrist, finger) in the real environment, which includes the number of accurately extracted human hand movement segments and the accuracy of the classification of human hand movement types. The second part is the recognition accuracy of human hand movements in the real environment, which is based on the accurate recognition of human hand movement types, and then the classification models (template library matching, wrist and finger minutiae decision tree model) for the hand movement types are selected to obtain the arm, wrist, and finger movement types. The system was therefore able to extract arm and wrist movement segments effectively.

4. Conclusions

The deep learning framework is a new teaching model, which is based on human cognitive structure and processes knowledge by certain means, and what needs attention in this process is how to systematize complex and abstract concepts into simple and clear. This paper constructs a wearable device built-in dance action recognition system through an improved deep learning algorithm, it can be found that the human hand action classification using decision tree in real environment has a high accuracy recognition rate, and the effect of human hand action-type classification is relatively ideal. It was caused by two main aspects: the amplitude of the movements was too small due to individual experimenters collecting finger data, as it was below the threshold of the algorithm causing finger movement segments to be filtered out. There was a relatively obvious jitter between two finger movements being recognized as one movement resulting in a lower number of action fragments than the true number of finger movements, and there were significant feature differences between the meticulous movements, resulting in the meticulous movements under the arm and wrist being effectively recognized by the classification model. However, the recognition accuracy rate of finger minutiae is not ideal. Analysis of it reveals that the magnitude caused by finger movement is not drastic and there is a certain difficulty in action fragment extraction, while there is a certain degree of similarity between finger minutiae movements, resulting in difficult recognition and low recognition rate. This shows that there is a role for wearable devices in the dance teaching affective mode of knowledge.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

References

[1] G. Chechik, I. Meilijson, and E. Ruppin, "Neuronal regulation: a mechanism for synaptic pruning during brain maturation [J]," *Neural Computation*, vol. 11, no. 8, pp. 2061–2080, 2019.

[2] F. I. Craik and E. Bialystok, "Cognition through the lifespan: mechanisms of change[J]," *Trends in Cognitive Sciences*, vol. 10, no. 3, pp. 131–138, 2016.

[3] R. C. Paolicelli, G. Bolasco, F. Pagani et al., "Synaptic pruning by microglia is necessary for normal brain development," *Science*, vol. 333, no. 6048, pp. 1456–1458, 2011.

[4] S. Elmore, "Apoptosis: a review of programmed cell death," *Toxicologic Pathology*, vol. 35, no. 4, pp. 495–516, 2007.

[5] P. Vanderhaeghen and H. J. Cheng, "Guidance molecules in axon pruning and cell death," *Cold Spring Harbor Perspectives in Biology*, vol. 2, no. 6, Article ID a001859, 2010.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet Classification with Deep Convolutional Neural networks[C]," in *Proceedings of the Wearable Computers, 2002. (ISWC 2002). Proceedings. Sixth International Symposium on*, pp. 1097–1105, NIPS, Lake Tahoe, 2012.

[7] G. Thimm and E. Fiesler, "High-order and multilayer perceptron initialization[J]," *IEEE Transactions on Neural Networks*, vol. 8, no. 2, pp. 349–359, 2017.

[8] D. Erhan, P. A. Manzagol, and Y. Bengio, *The Difficulty of Training Deep Architectures and the Effect of Unsupervised pretraining, Journal of Machine Learning Research*, vol. 5, pp. 153–160, 2009.

[9] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural networks," *in Proceedings of the 2nd International Conference on Learning Representations*, Banff, Canada, 2014.

[10] H. Sak, A. Senior, and F. Beaufays, *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic modeling*, INTERSPEECH, Singapore, pp. 338–342, 2014.

[11] A. H. Johnston and G. M. Weiss, "Smartwatch-based Biometric Gait recognition[C]," in *Proceedings of the IEEE, International Conference on Biometrics Theory, Applications and Systems*, pp. 1–6, IEEE, Arlington, VA, USA, 2015.

[12] V. M. Mantyla and J. Mantyjarvi, "Hand gesture recognition of a mobile device user[C]," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, vol. 1, pp. 281–284, IEEE, 2016.

[13] Z. He, L. Jin, and L. Zhen, "Gesture recognition based on 3D accelerometer for cell phones interaction[C]," in *Proceedings*

of the Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on, pp. 217–220, IEEE, 2008.

[14] T. Schlömer, B. Poppinga, and N. Henze, “Gesture recognition with a wii controller[C],” in *Proceedings of the International conference on tangible & embedded interaction*, pp. 11–14, ACM, 2008.

[15] J. Wu, G. Pan, D. Zhang, G. Qi, and S. Li, “Gesture recognition with a 3-D accelerometer,” in *Proceedings of the International Conference on Ubiquitous Intelligence and Computing*, vol. 5585, pp. 25–38, Springer-Verlag, 2009.

[16] S. J. Cho, E. S. Choi, and W. C. Bang, “Two-stage recognition of raw acceleration signals for 3-D gesture-understanding cell phones [C],” in *Proceedings of the The 10th International Workshop on Frontiers in Handwriting Recognition*, p. 6p. 6, Nijmegen, 2005.

[17] M. Shoaib, S. Bosch, and H. Scholten, “Towards detection of bad habits by fusing smartphone and smartwatch sensors[C],” in *Proceedings of the IEEE International Conference on Pervasive Computing and Communication Workshops*, pp. 591–596, IEEE, 2015.

[18] K. V Laerhoven, A. Schmidt, and H. W. Gellersen, “Multi-sensor context aware clothing[C],” in *Proceedings of the international symposium on wearable computers*, pp. 49–56, IEEE, 2002.